Boundary point method

An iterative algorithm to solve semidefinite programs

Mircea Simionica
Bocconi University
Milan, Italy
mircea.simionica@gmail.com

Contents

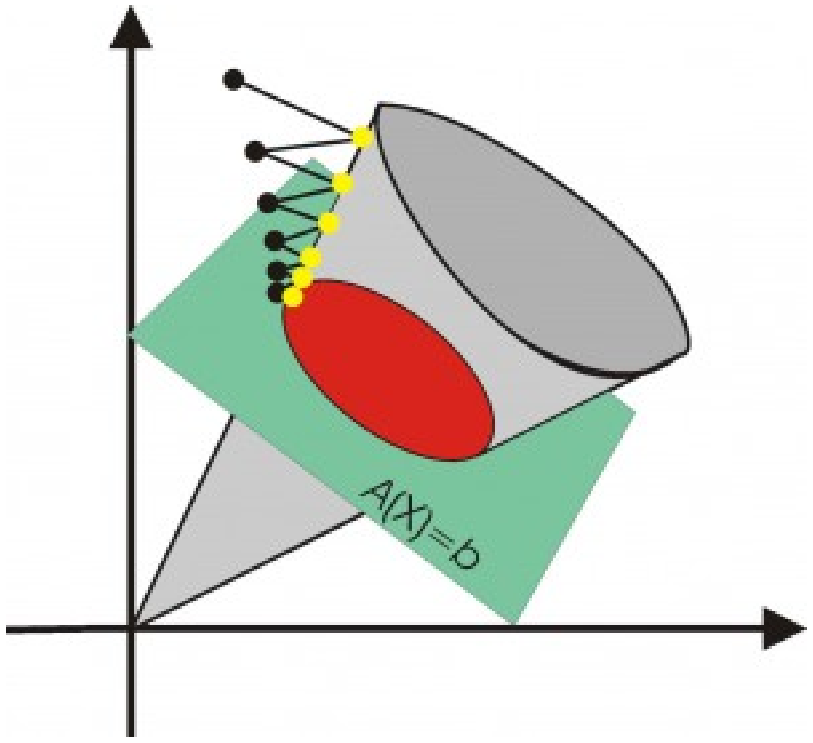

Figure 1: Illustration of the boundary point method.

1 Introduction

Welcome to the documentation for the BOUNDARY POINT METHOD code. This version was prepared for solving semidefinite programs with block diagonal structure. The code is written in commented C++ and available both in serial and in parallel version. In the latter, MPI is used as message-passing system. MATLAB is used for generating the data describing the block diagonal SDP. It is possible to run the .m scripts with Octave as well, freely available under the GNU license. To run the code you need to be on a *UNIX*-type system. Check section 3 for details on the required dependencies and external libraries.

2 The algorithm

The boundary point method (BPM) is an iterative algorithm for solving semidefinite programs (SDP). It has been developed by Povh, Rendl and Weigele. Although interior point methods (IPM) are the most prevalent method for solving SDPs, they are limited by memory requirements so problems with a large number of constraints are computationally expensive. BPM turned out to be efficient for problems where the number of constraints can be arbitrary. The algorithm considers the following primal-dual SDP:

$$(P) \max \langle C, X \rangle \text{ such that } A(X) = b,\ X \succeq 0,$$

$$(D) \min b^T y \text{ such that } A^T(y) - C = Z,\ Z \succeq 0,$$

with C and $A_i = 1, ..., m$ matrices of order n and b vector in $\mathbb{R}^m$.

The boundary point method exploits the Augmented Lagrangian technique and applies it to the dual formulation of the SDP. Letting X be a Lagrange multiplier for the dual equations $Z + C - A^T(y) = 0$ and for fixed $\sigma > 0$, the augmented Lagrangian L_σ looks as follows:

$$L_\sigma(y, Z; X) := b^T y + \left\langle X, Z + C - A^T(y) \right\rangle + \frac{\sigma}{2} \left\| Z + C - A^T(y) \right\|^2.$$

Let $W(y) := A^T(y) - C - \frac{1}{\sigma} X$. The Lagrangian can be rewritten as:

$$L_\sigma = b^T y + \frac{\sigma}{2} \left\| Z - W(y) \right\|^2 - \frac{1}{2\sigma} \left\| X \right\|^2.$$

Let $f(y, Z) := b^T y + \frac{\sigma}{2} \left\| Z - W(y) \right\|^2$. BPM minimizes $f(y, Z)$ iteratively, to get y and Z. Keeping Z constant, y is obtained from the linear system $AA^T(y) = A(Z + C + \frac{1}{\sigma} X) - \frac{1}{\sigma} b$. Then, while keeping y constant, Z is obtained from the eigen decomposition of W. These two steps represent the bottlenecks of the method. Then X is updated as $X \leftarrow X + \sigma(Z + C - A^T(y))$ and the process is iterated until convergence. The boundary point method was named after the fact that both Z and X lie on the boundary of the cone of semidefinite matrices. The complete algorithm is illustrated in *Figure 2*. For more details please refer to [1].

```
Select $\sigma > 0, \{\varepsilon_k\} \to 0,\ \varepsilon > 0$.
$k = 0;\ X^k = 0;\ Z^k = 0;$
repeat until $\delta_{outer} \leq \varepsilon$        ( Outer iteration for $k = 0, 1, \dots$ )
    repeat until $\delta_{inner} \leq \sigma\varepsilon_k$    ( Inner iteration: $(X^k,\ \sigma)$ held constant)
        Solve for $y^k$: $AA^T(y^k) = A(Z^k + C + \frac{1}{\sigma}X^k) - \frac{1}{\sigma}b;$
        $W = A^T(y^k) - C - \frac{1}{\sigma}X^k;$
        $Z^k = W_+;\ V^k = -\sigma W_-;$
        $\delta_{inner} = \|A(V^k) - b)\|;$
    end (repeat)
    $X^{k+1} = V^k;$
    $k \leftarrow k + 1;$
    $\delta_{outer} := \|Z^k - A^T(y^k) + C\|;$
end (outer repeat)
```

Figure 2: Boundary point method to solve the dual formulation.

3 Dependencies

The code is written in C++ and the parallel version makes use of MPI as message-passing system between processes. External linear algebra libraries are heavily used. Please find all the dependencies below: all of these should be already available on your system.

- C++ and MPI compilers with C++11 support.
- Armadillo: high quality C++ linear algebra library used throughout the whole code for matrix operations. It integrates with LAPACK and BLAS. Use Armadillo without installation by including the headers and linking against BLAS and LAPACK instead.
- Eigen: C++ template linear algebra library used for solving the sparse linear system. It has no dependencies, as it is defined in the headers.
- OpenBLAS: multi-threaded replacement of traditional BLAS. Recommended for significantly higher performance. Otherwise link with traditional BLAS and LAPACK.
- MATLAB/Octave: used for generating the data.

4 Block diagonal SDP

This version of the boundary point method was prepared for semidefinite programs with block diagonal structure. In the SDP formulation C and A matrices, as well as the solution X, are block diagonal matrices (check *Figure 3*). This kind of structure allows a parallelization dependent upon the number of diagonal blocks. While the serial code loops through the blocks, the parallel version allocates a block to each process and computations are done concurrently. MPI barriers are inserted at critical areas and results are collected on the root process at certain meeting points.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{A}_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \mathbf{A}_n \end{bmatrix}$$

Figure 3: Block diagonal matrix

5 Running the code

First make sure all the dependencies listed in section 3 have been met. To run the code you must first compile it. In the *Makefile* edit the entries regarding the location of headers and libraries. Then compile using

```
make
```

Change directory to either serial or parallel version. Copy an example of a SDP instance from the directory examples, for example

```
cp -r ../examples/8_blocks/data200 data
```

Run the serial code with the following command

```
./bpm
```

The parallel version is executed with

```
mpirun -np #blocks ./bpm
```

The solution is written in the *solution* directory.

5.1 Input data files

Input data files are located in *data* directory. The files are read at run time, so there is no need to recompile the code. Files containing sparse matrices resemble the compressed sparse column format.

file	description
dim	vector of dimensions of the blocks
AAT	m by m sparse matrix
A_GT$_i$	$dim(i)^2$ by m sparse matrix,
	where $dim(i)$ is the dimension of the i-th block
b	m by 1 vector
C$_i$	$dim(i)$ by $dim(i)$ sparse matrix of coefficients
nonzeros	vector of non-zero elements of all sparse matrices involved

Dense matrices are read using Armadillo's built-in function:

```
arma::mat A;
A.load(filename, raw_ascii);
```

Sparse matrices are constructed using batch insertions, which is faster than consecutively inserting values using element access operators.

5.2 Parameters

Parameters can be modified in *parameters/parameters.dat*. The file is read at run time.

parameter	description
sigma	penalty parameter in the Lagrangian function
tol	accuracy tolerance
max_iter	maximum number of outer iterations
in_max_init	initial number of inner iterations
in_max_inc	increase ratio of inner iterations at each outer iteration
in_max_max	maximum number of inner iterations
print_iter	defines output interval

References

[1] J. Povh, F. Rendl, A. Wiegele. A boundary point method to solve semidefinite programs, *Computing*, 78(3):277-286, November 2006.

[2] C. Sanderson Armadillo: an open source C++ linear algebra library for fast prototyping and computationally intensive experiments *Technical Report*, NICTA, 2010.

[3] Gaël Guennebaud and Benoît Jacob and others. *Eigen v3*, Retrieved from *http://eigen.tuxfamily.org*, 2010